2011 IEEE International Conference on High Performance Computing and Communications

An Evaluation of TCP and UDP Protocols Processing Required for Network Interface Design at 100 Gbps

Mohamed Elbeshti Michael Dixon Terry Koziniec
School of IT. Murdoch University, Perth, WA., Australia 6150
m.elbeshti@murdoch.edu.au

***Abstract*— Today major challenges are faced by server platforms while performing TCP/IP or UDP/IP protocol processing. For instance, the speed of networks now exceeds the gigabit per sec Gbps, the design and implementations of high-performance Network Interfaces (NI) have become very challenging. There are different possible design approaches to implement high-speed NI. However, using the General Purpose Processing (GPP) as a core engine to offload some, if not all, of the TCP/IP or UDP/IP protocol functions can deliver some important features to NIs, such as simplicity, scalability, shorter developing cycle time and reduced costs. Still it is not clear whether the GPP can provide the processing required for high-speed line over 10 Gbps. Also, where is the limit of such GPP in supporting the processing of network interfaces? In this paper, we have measured the amount of processing required for Ethernet Network Interfaces (ENI) design supporting different transmission line speeds. A NI-programmable based RISC model has been designed to measure the processing required for the ENI|. The results have shown that a RISC core running at 240 MHz can be used as a processing core in high-speed ENI. Such core can support a wide range of transmission line speeds, up to 100 Gbps. Also, we have discussed some of the design issues that are related to RISC core based NI and the data movement type.**

I. Introduction

There are number of challenges to design and implementing the ENI such as an absence of a standard NIC for any major OS. Another challenging task is to implement an algorithm in order to process a part of the protocol stack processing inside the NI, which is known as TCP Offload Engines (TOE) [11, 12, 14, 16, 22]. Also, the core engine in the NI needs to manage the out-of-order packets [1, 18, 26]. The NIs do support these functions but they would require better engine and good design in order to perform the mentioned functions. Beyond these challenges, the NIC must also support 10 Gbps or better (40 or 100 Gbps).

Generally, there are two possible methods that may be used to process the network interface protocols for TOE: First, designing a hardware to implement the use of either the Application Specific Integrated Circuit (ASIC) controller [10 ,15] or the use of Field Programmable Gate Arrays FPGA [6, 8]. Second method is Programmable-based NI that suggests the use of a General-purpose embedded processor [4, 5, 7, 19], or the use of specialized engine cores [14].

NIC Hardware-based

The technology advances in chip design, and specifically those which are ASIC based, have made it possible to integrate most, if not all, discrete components that are required for NI, on a single chip with low cost. Yet, the ASIC-based NIs have the difficulty in accepting new protocols without re-designing any single part of the chip components. FPGA platforms, on the other hand, are reconfigurable hardware with a specific NI function, which is more attractive to designers when building the NI, than the ASIC-based designs. Even though the FPGAs deliver a higher performance than ASIC-based, they still have a lot of set-up overheads compared to ASICs, which require more transistors to accomplish what an FPGA does with one. This adds latency, and increases the power consumption in the NI design. In addition, FPGAs become more sensitive to coding styles and design practices [25].

NIC Programmable-based

Designing the NI using the programmable method can potentially enhance the server performance [5], because the NIs programmable-base allows it more flexibility to adjust to the network protocol functions. These functions can be added or removed in the NI, simply by modifying the code of the protocol. The General-Purpose (GP) embedded processors, for instance, may not provide the same level of performance as the other methods offered, but they are more flexible, and can easily accommodate protocol revision or even new protocols. Moreover, the availability of the GPP contributes to the low development costs for network interfaces. Using these processors while designing the network interface simplifies the data path and, hence makes their design simple too.

The widely use of the hardware-based NIC such as the use of a fully customized logic based network interface can be due to the following reasons:

(a) There is no clear indication whether the processing speed offered by the GPP is fast enough to handle the NI functions such as Large Sending Offload (LSO) [24] function and data movement to match with the speed of the transmission line.

(b) As the new trend of designing the network interface is to have all the network interface functions implemented in one chip, the use of a commercially available GP embedded RISC core proves generally expensive and difficult to integrate within the network interface chip. Further, the size of these embedded RISC cores are large to be accommodated in the NI chip, since it is not designed for the NIs functions.

In this paper, we are investigating the amount of processing that is required by ENIs. In fact we are conducting a research at Murdoch University to design a specialized RISC core for ENI for programmable networks, and the measurements that have been presented in this paper are part of this research.

978-0-7695-4538-7/11 $26.00 © 2011 IEEE
DOI 10.1109/HPCC.2011.173

The rest of this paper is organized as follows: Section 2 will investigate the model that we have designed in our simulation. The simulation results are discussed in the next two sections (3 and 4). Section 5 discusses the RISC core structures followed by a conclusion in the last section.

II. ENI MODEL

To measure the processing required for high speed NIs, a NI model has been proposed. The architecture of the NI is commonly partitioned into three parts: the communication line interface, the processing core, and the host bus interface (Figure 1). The line and host interfaces are implemented in hardware. The processing cores perform commonly processed functions for send side such as LSO; and for receive side such as Large Receive Offload (LRO) [2, 3, 9, 13]. The IP address and port ID support out-of-order packets, data movements due to packet copying and linked-list mechanism.

The NI model shown in Figure 1 processes all the above mentioned functions. The processing core of this model has been evaluated by using both TCP/IP and UDP/IP. Other protocols can be evaluated in this model, but are not included in this work because of the wide use of the applications over the TCP or UDP protocol and the time constraints of this research.

During the research it was observed that as the packets arrive into the Receiver Buffer Interface (RBI), the RISC core at the receiving side will be interrupted. The RISC core will then start processing the packet headers and identify the type of packet by reading the protocol type inside the IP header and TCP or UDP header. The fields inside the TCP/IP (such as the IP address inside the IP header and the ID and sequence number inside the TCP header) are used to check whether the packet belongs to an existing stream that has already been amalgamated in the Receiver Buffer (RB) or if it is the first packet of a new stream. In the UDP/IP, the IP address and the port ID are responsible for recognizing the packet if it is related to a UDP stream, while the Offset and the ID fields inside the IP header are used to identify whether the packet is the starting packet, the continuation or final packet of the stream. A linked-list mechanism is also processed for the incoming packets, where every packet is linked with the previous stream of the same connection and buffered inside the RB. A memory management state machine is responsible for providing free available spaces that exist on the RB to the RISC core. The free pointers occurrences are collected after the host reads the amalgamated data from the RB and the same are used for newly arrived packets. After the headers have been processed by RISC core, the data movement operation moves the body of the packet and the payload part towards the RB. Three First-in-First-outs (FIFO) are used in the receiver unit. The first FIFO in this model is used to store the control signaling messages that a host uses to establish a new connection with the other end, or for flow control messages. The second FIFO is used to hold the pointers for amalgamated packets. The amalgamated pointers are to be inserted in the FIFO whenever the Interpret Moderation (IM) time expires [17].

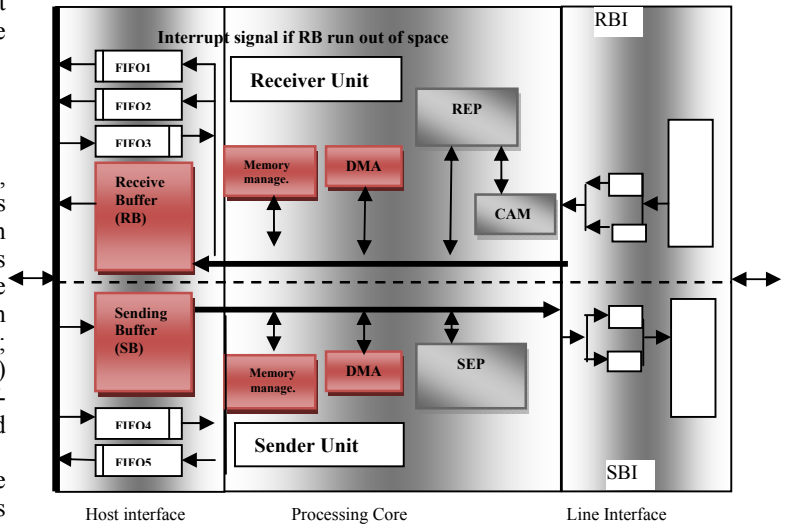

Figure 1. The Ethernet Netwrok Interface (ENI) model

The host then uses these pointers to move the amalgamated data to the host memory. Finally, the TCP/IP active connections information is sent by the host CPU thro the FIFO3 to the reassembly unit in the NI. Con Addressable Memory (CAM) is also used with receiver unit's RISC core to help in efficiently processir linked list processing and to keep the status of all the NI connections.

When the host sends a large frame (over the MTU) to the Sending Buffer (SB), the RISC core then cuts the large frame to small pieces (fit into MSS), then generates the headers chnique is required in this stage, by initiating the DMA inside the sender unit to move the payload part of the packet to Sending Buffer Interface (SBI), while the header is moved towards SBI. The SBI then sends a complete frame over the transmission line. The sender unit has only two FIFOs. The first is used to hold the signaling packets that are related to a new connection with the other end. Also, the FIFO carries the small packets, less than the MTU that the host needs to send to a network [24]. The second FIFO sends the free pointer inside the SB that is prepared by the memory management after the RISC completes sending a message. Also, the memory management engine informs the RISC core in the sender side while the TCP or UDP data are located inside the SB.

III. THE SIMULATION

The proposed model was implemented using the SPIM simulator [20, 21] with Million Instructions Per Second (MIPS) R2000/R3000 processor. All the TCP/IP and UDP/IP functions that have been mentioned in this study are simulated, in addition to data movement. The simulation for the sending unit is completely independent

of the simulation for the receiving side, since in this model a processor shall be used for each function to improve performance. Both sides are processed in parallel and each one has its own data path to transfer data to/from NI buffers. The data movement for the packet payload is simulated using the DMA mechanism. The RISC needs less MIPS when the DMA mechanism is used instead of using the programmed I/O for data movements, especially when the payload size is 1460 (the MSS See Figure 6). However, a small portion of data using programmable I/O method is more practical than initiating the DMA to move small part of data (see Figure 7.) Using the DMA method with MSS, for example, the RISC core will be free while the DMA performs the data transfer to/from Line Interface. During data movement the RISC core at the sending or the receiving side may perform other activities that are not related to the NI's bus such as generating the IP, TCP or UDP header. In the receiving side the RISC core is also free during the data movements. It updates the linked-list inside the CAM's data or inserts a new set of CAM entries.

IV. SIMULATION RESULTS

The simulator has measured the amounts of processing that are required for TCP/IP and UDP/IP protocol processing and for data movement. Different TCP/IP and UDP/IP packets have been delivered to the simulator and the number of processed instructions required for the protocols, with and without data movement processing, were measured in MIPS, where every instruction was processed in one cycle. Therefore, the results shown below represent the required speed of the RISC core in terms of MHz while processing the requirements for the amalgamated function alone (without data movement) for both TCP/IP and UDP/IP. In case of the upper bound processing for TCP/IP and UDP/IP, the results are shown in Figure 2 which represents the maximum Receive Embedded Processor (REP) clock rate to process different transmission lines when the packet size is 1460 B. In Figure 2, the REP clock is rated when the small size packet is applied. It is clear that the clock of the RISC core gets higher while performing the small size packets. This is obvious because about 14,880.952 packets arrive in one second when the line is 10 Gbps comparing to large packets (1460) 812.744 packets.

Figure 4 shows the amount of processing that the RISC core needs to process the LSO function and to initiate the DMA controller. From the result it can be seen that, once the transmission lines speed get higher (over 10 Gbps) the amount of processing to handle the LSO and initialization of the DMA controller becomes significant. Figure 5 depicts that the SEP clock is rated when a small size packet is applied. In our results UDP uses less MIPS when performing the LSO and RSA. This is a consequence of the reduced processing requirements for UDP/IP when compared to TCP/IP. UDP operates on a best-effort basis and leaves the other functions associated with end-to-end reliable transport to higher layers of the protocol. On the other hand, TCP provides end-to-end reliable transmission.

V. THE TARGET RISC CORE

The simulation results demonstrated that a RISC-based NI is scalable for a transmission line with a speed up to 100 Gbps. To reduce the design complexity, we presented a simple data path of the NI. This simplicity helps the RISC cores to manage and process the RSA and LSO at a low clock rate. Further it has made it possible to reduce the cost of development of RISC-based NIs. Such NIs can be flexible enough to support protocol changes or can even adapt new protocols, whereas customized logic-based NIs can only support specific functions. The design of a RISC core for specialized application, namely NI control and data path, is simpler than that of GP processors because the general-purpose embedded processors is not optimized for specific protocol application. Hence, some portions of GP instructions that support general-purpose applications may not be required for the ENI design. For example, the Floating-Point Unit is not necessary for network interfaces. Also, we found that, using a data cache to store data [3] is not required since it will not enhance the NI's performance or reduce the RISC' clock for this application. The elimination of these units in a core makes it simpler to develop and reduce its cost.

We have noticed from the simulation processing that programs executed by RISC core use fewer types of instructions for processing the NI's functions. These instructions that are required for LSO are load, store, arithmetic and logic operation and conditional branches (see Table 1) Also, we measured the total percentage of each type of these instructions that the RISC core is required to perform the LSO. For RSA it was discovered that the types of instructions are load, store, arithmetic and logic operation and conditional branches (See Table 2). It is hence concluded that minimum instructions set can be used with the core, which would make the control unit design very simple and fast. In addition, the limited number of instructions that are required to support the Ethernet interface processing can reduce the size and complexity of the control unit leading to an increased speed.

TABLE 1. Type of instruction the RISC needs for LSO function

Operation type	**TCP/IP** Processing	**UDP/IP** Percentage rate
Load	22%	23%
Store	33%	35%
Arithmetic and logic operation	27%	29%
Conditional branch	16%	11%
Reading/writing from/to HNIC	28%	29%
The LSO data structure	72 %	71%

TABLE 2.Type on instruction the RISC needs for RSA functions

Operation type	**TCP/IP** Processing	**UDP/IP** Percentage rate
Load	39.28 %	40 %
Store	17.8 %	24.0 %
Arithmetic and logic operation	28.27 %	24.0 %
Conditional branch	14.28 %	12.0 %
Reading/writing from/to HNIC	26.9 %	26.9%
The RSA data structure	40.81 %	39.8 %

VI. CONCLUSION

We have presented computer simulations results to measure the amount of processing required for both TCP/IP and UDP/IP. The simulation results have shown that a cost effective embedded RISC core can efficiently provide network interface with the processing that is required supporting a wide range of transmission line speed. A 239 MHz RISC core can support the Receiver unit processing for up to 100 Gbps transmission speed for TCP/IP, while a core running at 214 MHz is found to support UDP/IP protocol when the MTU is applied (1500 B). For sending side, a 154 MHz is found to support line speed up to 100 Gbps when TCP/IP is used and 145 MHz for UDP/IP. These research results will play an important role in the next generation of business and educations applications such as e_Learning which will require faster processing.

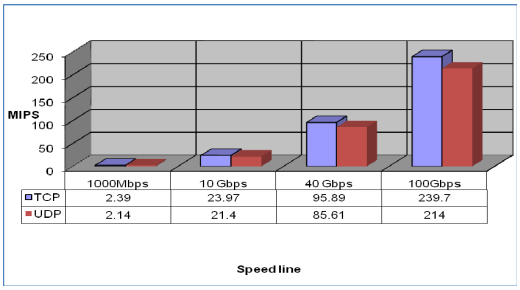

Figure 2: RISC clock rate at receive uint when the packet size is (MTU) 1500 byts using DMA

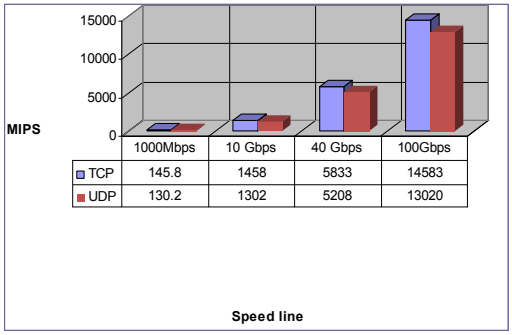

Figure 3. RISC clock rate at the reciver unit when the packet size is the smallest size (64 bytes) using DMA

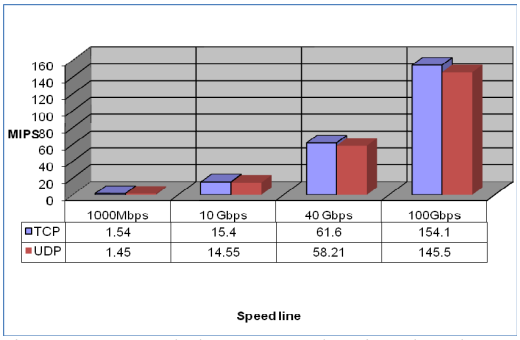

Figure 4. RISC clock rate at sender uint when the packet size is (MTU) 1500 bypts using DMA

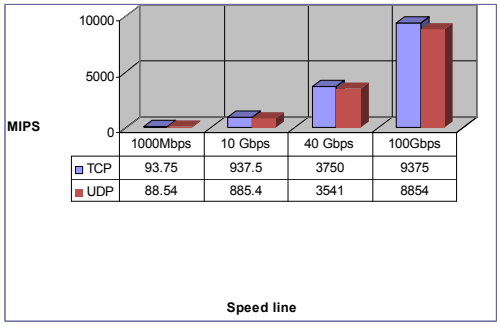

Figure 5. RISC clock rate at Sender unit when the packet size is smallest size (64 bypts) using DMA

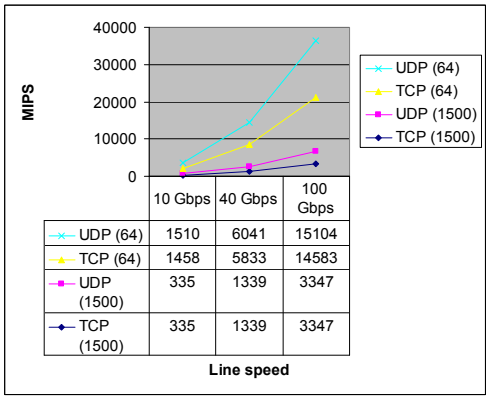

Figure 6. Receiver side with data movemt unit using programble I/O

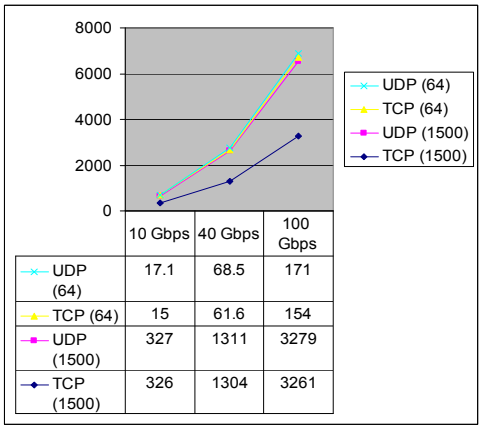

Figure 7. Sender side with data movemt unit using programble I/O

REFERENCES

[1] G. Held. "Ethernet Networks (4th ed.)," Design, Implantation, Operation and Management. John Wiley publisher LTD, 2003.

[2] Makineni, S., Iyer, R., Sarangam, P., Newell, D., Zhao, L., Illikkal, R., Moses, J. "Receive Side Coalescing for Accelerating TCP/IP Processing," In: Robert, Y., Parashar, M., Badrinath,R., Prasanna, V.K. (eds.) HiPC 2006. LNCS, vol. 4297, pp. 289-300. Springer, Heidelberg(2006).

[3] P. Govindarajan et al. "Achieving 10Gbps Network Processing: Are We There Yet?" High Performance Computing – HiPC. Pp 518-528 2008.

[4] P. Willmann, K. Hyong-youb, S. Rixner and S. Pai, "An Efficient Programmable 10 Gigabit Ethernet Network Interface Card," Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture November 2005.
[5] H. Kim, "Improving Networking Server Performance with Programmable Network Interfaces," RICE University, Master's thesis, April, 2003.
[6] T. Mohsenin, "Design and Evaluation of FPGA-Based Gigabit-Ethernet/PCI Network Interface Card," Rice University, Master's thesis, April 2004.
[7] X. Yang, D. Wu and N. Sun." Design of NIC *Based on I/O Processor for* Cluster Interconnect Network," Networking, Architecture, and Storage, pp 3-8 July 2007.
[8] T. Henriksson, " Intra-Packet Data-Flow Protocol Processing," PhD Dissertation, Linkoping university,2003.
[9] A. Menon and W. Zwaenepoel. Optimizing TCP receive performance. In USENIX Annual Technical Conference, June 2008.
[10] O. Elkeelany, "On chip novel video streaming system for bi-network," multicasting protocols, Integration, the VLSI Journal, v.42 n.3, p.356-366, June, 2009.
[11] *Alacritech SLIC:" A Data Path TCP Offload methodology,*" *http://www.alacritech.com/html/techreview.html*
[12] Y. Hoskote, et al., "A 10 GHz TCP Offload Accelerator for 10 Gb/s Ethernet in 90nm Dual-Vt CMOS," IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, 2003.
[13] L. Grossman. "Large Receive Offload implementation in Neterion 10GbE Ethernet driver". In *Ottawa Linux Symposium (OLS)*, 2005.
[14] Earls, "*TCP Offload Engines Finally Arrive,*" Storage Magazine, March 2002.
[15] Y. Hoskote et al., "A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS", *IEEE Journal of Solid-State Circuits*, 38(11):1866–1875, Nov. 2003.
[16] K. Kant "TCP offload performance for font-end server" Proc,IEEE Global telecommunications conference (GLOBECOM 03) , IEEE press, 2003 pp 3242-3247**.**
[17] Interrupt Moderation Using Intel® GbE Controllers 2007. *download.intel.com/design/network/applnotsap450.pdf.*
[18] J. B. Postel, "Transmission Control Protocol TCP," RFC 793, Information Sciences Institute, Sept. 1981.
[19] S. Pai and S. Rixner, "Exploiting task-level Concurrency in a Programmable Network Interface," Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp 61-72, 2003.
[20] " A MIPS32 Simulator 2010"**.** http://pages.cs.wisc.edu/~larus/spim.html
[21] D. Patterson and J. Hennessy," Computer Organization and Design," The Hardware/Software Interface. Morgan Kaufmann, Los Altos, CA 1998.
[22] S. Senapathi and R. Hernandez," TCP Offload Engines," Network AND Communications magazine pp103-107 , 2004 .
[23] J. Mogul, "TCP Offload Is a Dumb Idea Whose Time Has Come," *Proc. 9th Workshop on Hot Topics in Operating Systems* (HotOS IX), Usenix Assoc., 2003; www.usenix.org/events/hotos03/tech/full_papers/mogul/mogul.pdf.
[24] Cardona et al. " System Load Based Dynamic Segmentation for Network Interface Cards." Patent US20080295098. **Nov. 27,** 2008
[25] Xiling company "Xilinx Design Reuse Methodology for ASIC and FPGA Designers" 2010 http://www.fpga.com.cn/advance/skill/Design_Reuse_Methodology.pdf.
[26] M. Degermark and B. Nordgren "IP Header Compression", RFC 2507 Network Working Group 1999